

# Artificial Intelligence

CS 165A

May 17, 2022

Instructor: Prof. Yu-Xiang Wang

T  
O  
D  
A  
Y

- Contextual Bandits
- Reinforcement Learning

# Recap: Multi-arm bandits: Problem setup

- No state. k-actions  $a \in \mathcal{A} = \{1, 2, \dots, k\}$

- You decide which arm to pull in every iteration

$$A_1, A_2, \dots, A_T$$

- You collect a cumulative payoff of  $\left[ \sum_{t=1}^T R_t \right]$  —  $\min \sum_{t=1}^T \mathbb{E}[R_t]$   
 $\left[ \sum_{t=1}^T R_t \right]$  —  $\min \sum_{t=1}^T \mathbb{E}[R_t]$   
 $\left[ \sum_{t=1}^T R_t \right]$  —  $\min \sum_{t=1}^T \mathbb{E}[R_t]$
- The goal of the agent is to maximize the expected payoff. —  $\text{regret}$ 
  - For future payoffs?
  - For the expected cumulative payoff?  $= o(T)$

# Announcements

- Project 2
  - Most of you did very well! Excellent job!
  - You may still submit your solutions for bonus part.
- Project 3 announced.
  - Please start early!
  - The MDP part (Q1, Q2, Q3) of it is already covered.
  - The RL part (Q6 onwards )will be fully covered by the end of the week.

# Recap: How do we measure the performance of an **online learning agent**?

- The notion of “Regret”:
  - I wish I have done things differently.
  - Comparing to the best actions in the hindsight, how much worse did I do.
  
- For MAB, the regret is defined as follow

(Cumulative)  $\Rightarrow$   $T \max_{a \in [k]} \mathbb{E}[R_t | a] - \sum_{t=1}^T \mathbb{E}_{a \sim \pi} [\mathbb{E}[R_t | a]] = O(T^\alpha)$   
(Expected) Regret  $\alpha < 1$

Avg Regret  $\rightarrow 0$   
 $T \rightarrow \infty$

# Recap: MAB Algorithms

- Idea: Plug-in estimate of the reward value
- Greedy:  $\text{Regret} = O(T)$
- Explore-first:  $\text{Regret} = O(T^{2/3})$
- epsilon-greedy:  $\text{Regret} = O(T^{2/3})$
- Upper Confidence Bound:  $\text{Regret} = O(T^{1/2})$ 
  - Optimal in the sense that no algorithm can do better

# Recap: Upper Confidence Bound algorithm (UCB)

- At time  $t$ , choose the action

$$A_t \leftarrow \operatorname{argmax}_a \left[ \underbrace{Q_t(a)}_{\text{plugin}} + c \underbrace{\sqrt{\frac{\log(1+t)}{N_t(a)}}}_{\text{Bonus}} \right]$$

*uncertainty estimator* (pointing to the square root term)  
*error estimate* (pointing to the square root term)  
 $O(\frac{1}{\sqrt{n}})$  (next to the square root term)

- Idea: Be optimistic
  - Choose an option that maximizes the upper confidence bound.

$$\mathbb{E}[\text{Regret}] = O(\sqrt{Tk})$$

- The proof is out of the scope of this course. For those who are interested, please look up. It's not difficult.

# Idea of the analysis of UCB

$$Q_t(a) \pm \epsilon \quad \epsilon = O\left(\sqrt{\frac{1}{N_t(a)}}\right)$$

- Design principle: Optimistic in the face of uncertainty
- Idea why UCB improves over random exploration:
  - When you follow the UCB approach, the maximum regret that you can incur in each iteration is the confidence interval of the arm you pick. (why is that?)
  - Exploration will be restricted to those arms that are not "eliminated" yet.

$$E[\text{Regret}_t] = Q(a^*) - Q(\hat{a}) \leq Q_t(a^*) + \epsilon - (Q_t(\hat{a}) - \epsilon)$$

$$\hat{a} = \underset{a}{\text{argmax}} Q_t(a) + \epsilon$$

$$\leq Q_t(a^*) + \epsilon - (Q_t(\hat{a}) - \epsilon) = Q_t(a^*) + \epsilon - Q_t(\hat{a}) + \epsilon$$

- In other words, UCB explore and exploit at the same time!

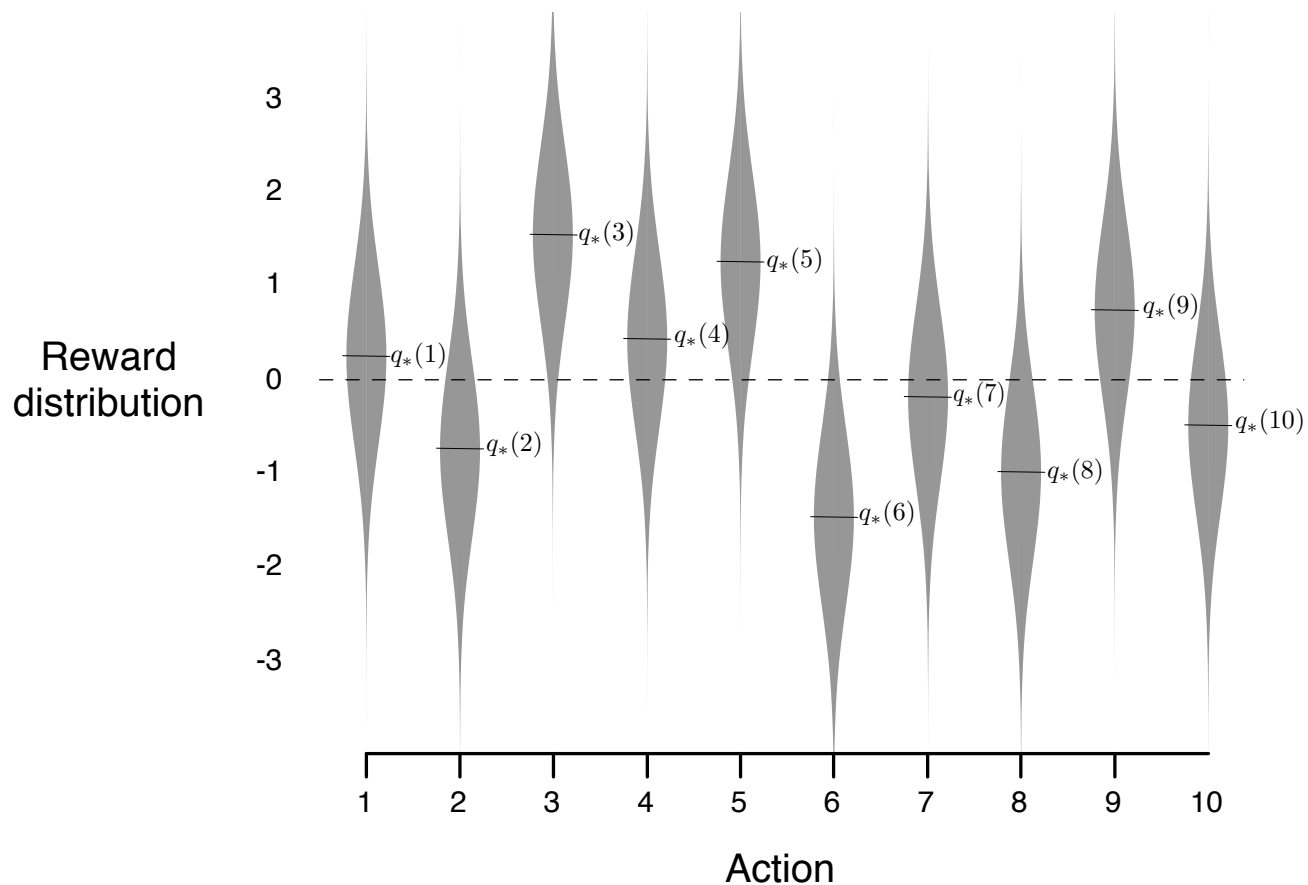
$$\leq 2 \cdot \epsilon \approx O\left(\frac{1}{\sqrt{N_t(a^*)}}\right)$$

# Intuitively why is the $O(T^{1/2})$ regret optimal?

- Consider a 2-arm bandit problem and two parallel worlds:
  - Arm 1 has expected reward 0.5, Arm 2 has  $0.5 + \epsilon$
  - Arm 1 has expected reward  $0.5 + \epsilon$ , Arm 2 has 0.5
  - Reward distribution is Bernoulli distribution.
- Set  $\epsilon = O(1/\sqrt{T})$ . Recall that you need to pull Arm 1 and Arm 2 both for  $\Omega(T)$  times in order to identify which one is better. Thus the regret needs to be  $\Omega(T \times \frac{1}{\sqrt{T}})$ .
- To say it differently: If any algorithm is able to achieve better regret, then it implies an estimator that estimates the  $p$  of a biased coin with fewer samples than required. Thus a contradiction.

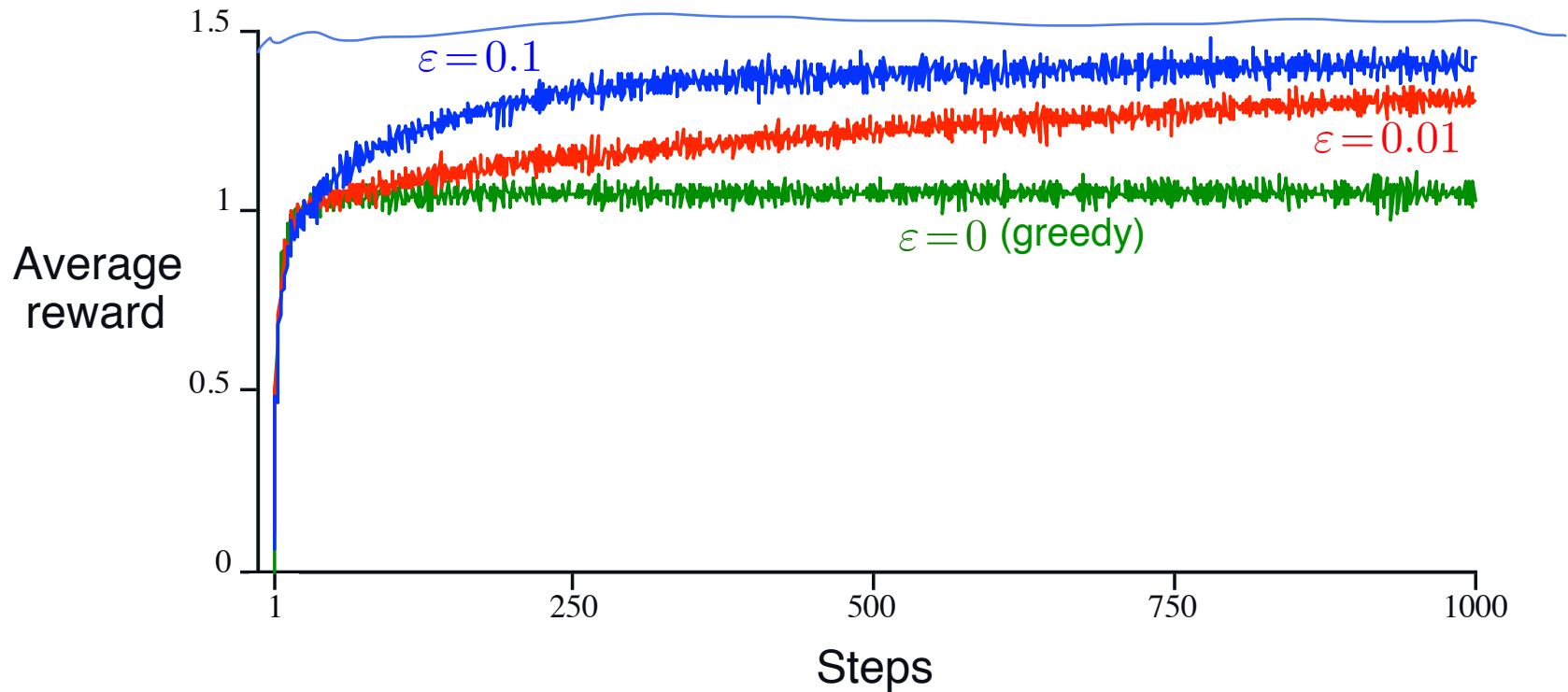


# A 10-armed bandits benchmark

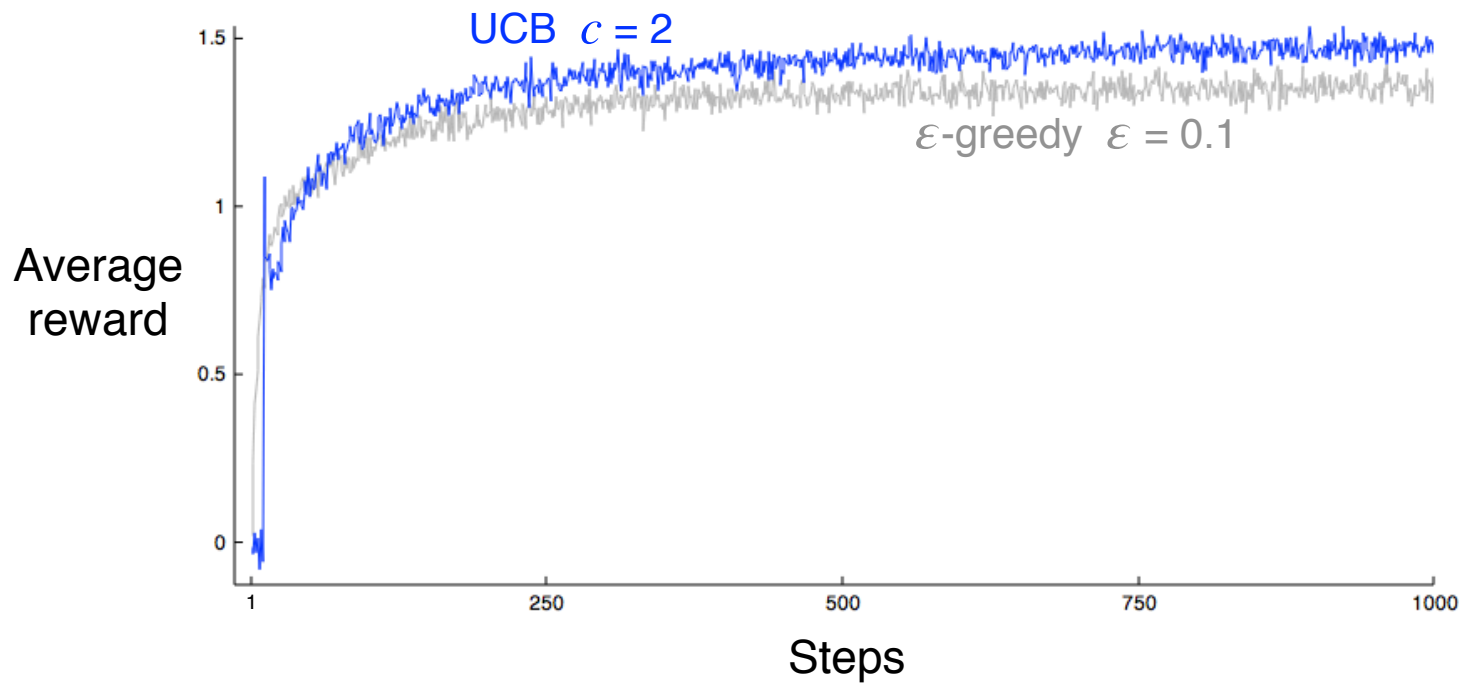


**Figure 2.1:** An example bandit problem from the 10-armed testbed. The true value  $q_*(a)$  of each of the ten actions was selected according to a normal distribution with mean zero and unit variance, and then the actual rewards were selected according to a mean  $q_*(a)$  unit variance normal distribution, as suggested by these gray distributions.

# Comparing the different algorithms



# UCB vs. $\epsilon$ -Greedy



# Variants of Bandits problems

- Online Learning from Expert Advice
  - Adversarial chooses the outcome
  - You observe outcome of other arms as well
  - Compare against the best arm in the hindsight
- Adversarial  $k$ -Armed Bandits
  - Same as above. But you observe only your arm.
- Nonstationary Bandits
  - Stochastic but the reward distribution changes over time.
  - Compare against the best arm for each time.
- Contextual bandits: you have a state in each time point.

# Variants of Bandits problems

- Online Learning from Expert Advice
  - Adversarial chooses the outcome
  - You observe outcome of other arms as well
  - Compare against the best arm in the hindsight
- Adversarial k-Armed Bandits
  - Same as above. But you observe only your arm.
- Nonstationary Bandits
  - Stochastic but the reward distribution changes over time.
  - Compare against the best arm for each time.
- Contextual bandits: you have a state in each time point.

**Remark:** In all these problems, there are algorithms with provably low-regret.

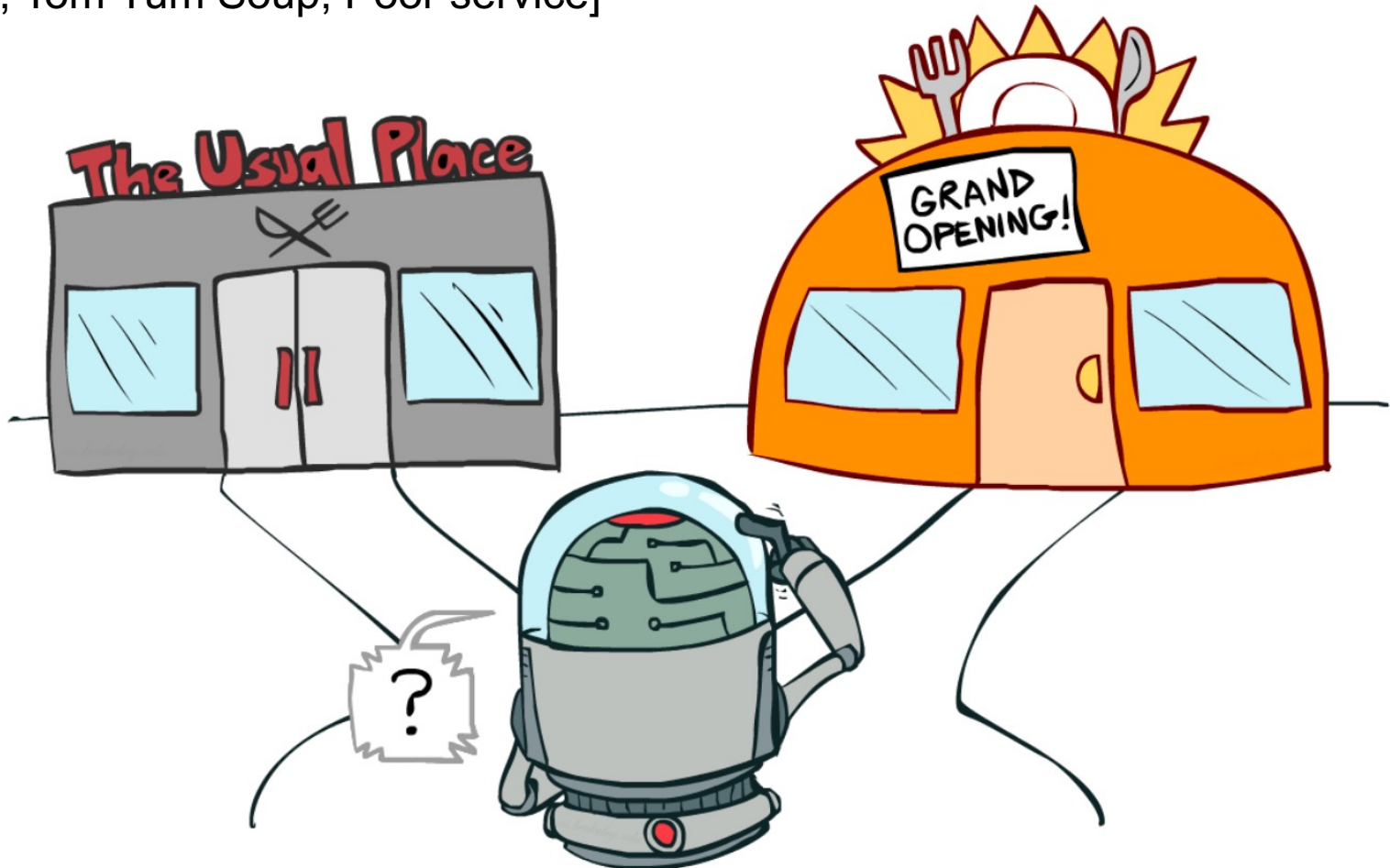
# Do I have to try, if I have features?

Features:

[Noodles, Tom Yum Soup, Poor service]

Features:

[Burger, Fries, Onion Ring, Fried Chicken]



(Illustration from Dan Klein and Pieter Abbeel's course in UC Berkeley)

# We know how to use with features, don't we?

- Classifier agent
  - Take features of a restaurant as input
  - Output a prediction of “will I like the food?”
- Train with supervised learning
  - Using the my previous visits to the restaurants
  - Using Yelp reviews

# We know how to use with features, don't we?

- Classifier agent
  - Take features of a restaurant as input
  - Output a prediction of “will I like the food?”
- Train with supervised learning
  - Using the my previous visits to the restaurants
  - Using Yelp reviews

Why can't we just use that?



# We know how to use with features, don't we?

- Classifier agent
  - Take features of a restaurant as input
  - Output a prediction of “will I like the food?”
- Train with supervised learning
  - Using the my previous visits to the restaurants
  - Using Yelp reviews

Why can't we just use that?

How to explore?

# Contextual Bandits: Problem Setup

- For each round  $t = 1, 2, 3, \dots, T$ :
  - A context  $x_t \sim$  unknown distribution i.i.d.
  - Agent picks an action  $a_t = 1, 2, 3, \dots, K$
  - Reward  $r_t \sim D(\cdot | x_t, a_t)$

- Agent's goals:

A finite family of policies

- Learn the best policy out of many policies  $\Pi$
- Minimize the cumulative regret

$$T \cdot \max_{\pi \in \Pi} \mathbb{E}_{\pi} [r_t(x_t, a_t)] - \mathbb{E}_{\text{Agent's policy}} \left[ \sum_{t=1}^T r_t(x_t, a_t) \right]$$

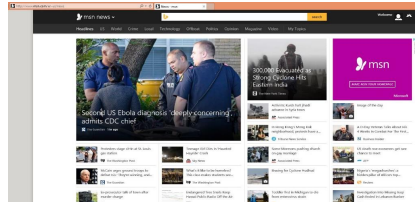
Reward from the best policy

Reward collected by the Agent

# Applications of Contextual Bandits

# Applications of Contextual Bandits

Personalized news?



Repeatedly:

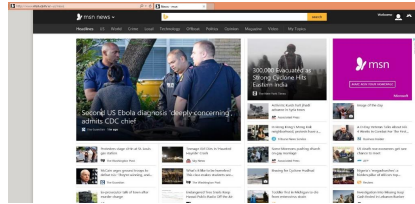
1. Observe features of user+articles
2. Choose a news article.
3. Observe click-or-not

$$X_t = (\text{feature of user}, \text{feature of article})$$

**Goal: Maximize fraction of clicks**

# Applications of Contextual Bandits

Personalized news?

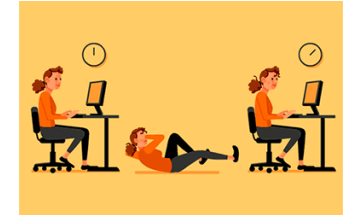


Repeatedly:

1. Observe features of user+articles
2. Choose a news article.
3. Observe click-or-not

**Goal: Maximize fraction of clicks**

Health advice?



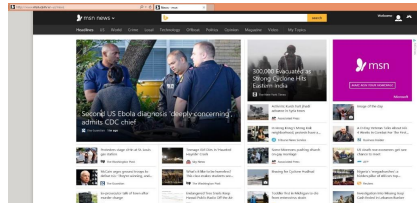
Repeatedly:

1. Observe features of user+advice
2. Choose an advice.
3. Observe steps walked

**Goal: Healthy behaviors (e.g. step count)**

# Applications of Contextual Bandits

Personalized news?

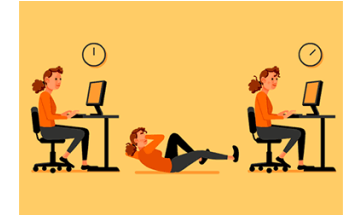


Repeatedly:

1. Observe features of user+articles
2. Choose a news article.
3. Observe click-or-not

**Goal: Maximize fraction of clicks**

Health advice?



Repeatedly:

1. Observe features of user+advice
2. Choose an advice.
3. Observe steps walked

**Goal: Healthy behaviors (e.g. step count)**

The Amazon logo, consisting of the word "amazon" in a bold, black, sans-serif font with a curved orange arrow underneath it.

Recommendations



buy or not buy



# Exploration vs. Exploitation in Contextual Bandits.

# Exploration vs. Exploitation in Contextual Bandits.

- Challenging because:
  - **Infinite state space**, never see the same context again.
  - **Exponentially large** policy space



# Exploration vs. Exploitation in Contextual Bandits.

- Challenging because:
  - **Infinite state space**, never see the same context again.
  - **Exponentially large** policy space
- Ideas:
  - ExploreFirst,  $\epsilon$ -Greedy  $O(T^{2/3})$

# Exploration vs. Exploitation in Contextual Bandits.

- Challenging because:
  - **Infinite state space**, never see the same context again.
  - **Exponentially large** policy space
- Ideas:
  - ExploreFirst,  $\epsilon$ -Greedy  $O(T^{2/3})$
  - UCB? But how do we construct Confidence Interval for an exponentially large set of policies?

# Exploration vs. Exploitation in Contextual Bandits.

- Challenging because:
  - **Infinite state space**, never see the same context again.
  - **Exponentially large** policy space
- Ideas:
  - ExploreFirst,  $\epsilon$ -Greedy  $O(T^{2/3})$
  - UCB? But how do we construct Confidence Interval for an exponentially large set of policies?

- Optimal regret:

$$O(\sqrt{KT \log |\Pi|})$$

*(Handwritten blue annotations: an underline under  $\log |\Pi|$ , an arrow pointing to  $K$ , and a checkmark under  $T$ )*

# Remainder of the lecture today

- Reinforcement learning for MDPs
  - Model-based vs model-free algorithms
  - Online policy iterations
  - Temporal difference learning
- Readings:
  - AIMA Ch. 21.1-21.3 (Ch 22.1- 22.3 in 4th Edition)
  - Sutton and Barto: Ch 4-6
  - Maybe: Sutton and Barto: Ch 6, Ch 13

# Let us tackle different aspects of the RL problem one at a time

- Markov Decision Processes:
  - Dynamics are given no need to learn
- Bandits: Explore-Exploit in simple settings
  - RL without dynamics
- **Full Reinforcement Learning**
  - Learning MDPs

# Recap: Tabular MDP

- **Discrete** State, **Discrete** Action, Reward and Observation

$$S_t \in \mathcal{S} \quad A_t \in \mathcal{A} \quad R_t \in \mathbb{R} \quad \text{---} \quad \cancel{O_t \in \mathcal{O}}$$

- Policy:

– When the state is observable:  $\pi : \mathcal{S} \rightarrow \mathcal{A}$

~~– Or when the state is not observable~~

$$\text{---} \quad \cancel{\pi_t : (\mathcal{O} \times \mathcal{A} \times \mathbb{R})^{t-1} \rightarrow \mathcal{A}}$$

- Learn the best policy that maximizes the expected reward

– Finite horizon (episodic) RL:  $\underline{\pi}^* = \arg \max_{\pi \in \Pi} \mathbb{E} \left[ \sum_{t=1}^T R_t \right]$  **T: horizon**

– Infinite horizon RL:  $\underline{\pi}^* = \arg \max_{\pi \in \Pi} \mathbb{E} \left[ \sum_{t=1}^{\infty} \gamma^{t-1} R_t \right]$

**$\gamma$ : discount factor**

# Recap: Policy Iterations and Value Iterations

- What are these algorithms for?
  - Algorithms of computing the  $V^*$  and  $Q^*$  functions from MDP parameters

- Policy Iterations

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

- Value iterations

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s, a) [r(s, a, s') + \gamma V_k(s')]$$

- How do we make sense of them?
  - Recursively applying the Bellman equations until convergence.

# Recap: Policy Iterations and Value Iterations

- What are these algorithms for?
  - Algorithms of computing the  $V^*$  and  $Q^*$  functions from MDP parameters

- Policy Iterations

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

- Value iterations

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s, a) [r(s, a, s') + \gamma V_k(s')]$$

- How do we make sense of them?
  - Recursively applying the Bellman equations until convergence.

\*These methods are called “Dynamic Programming” approaches in Chap 4 of Sutton and Barto.



# Revisit the dynamic programming approach

- Policy Evaluation

$$V_{k+1}^{\pi}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} P(s'|s, a) [r(s, a, s') + \gamma V_k^{\pi}(s')]$$

- Policy improvement

$$\begin{aligned} \pi'(s) &= \arg \max_a Q^{\pi}(s, a) \\ &= \arg \max_a \sum_{s'} P(s'|s, a) [r(s, a, s') + \gamma V_k^{\pi}(s')] \end{aligned}$$

- Value iterations

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s, a) [r(s, a, s') + \gamma V_k(s')]$$

# Revisit the dynamic programming approach

- Policy Evaluation

$$V_{k+1}^{\pi}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \cancel{P(s'|s, a)} [\cancel{r(s, a, s')} + \gamma V_k^{\pi}(s')]$$

- Policy improvement

$$\begin{aligned} \pi'(s) &= \arg \max_a Q^{\pi}(s, a) \\ &= \arg \max_a \sum_{s'} \cancel{P(s'|s, a)} [\cancel{r(s, a, s')} + \gamma V_k^{\pi}(s')] \end{aligned}$$

- Value iterations

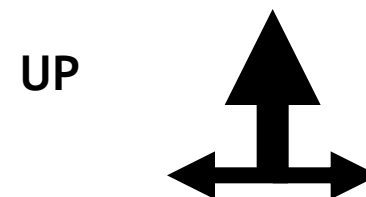
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} \cancel{P(s'|s, a)} [\cancel{r(s, a, s')} + \gamma V_k(s')]$$

**\*We do not have the MDP parameters in RL!**

# Example: Frozen Lake

			+1
			-1
START			

actions: UP, DOWN, LEFT, RIGHT



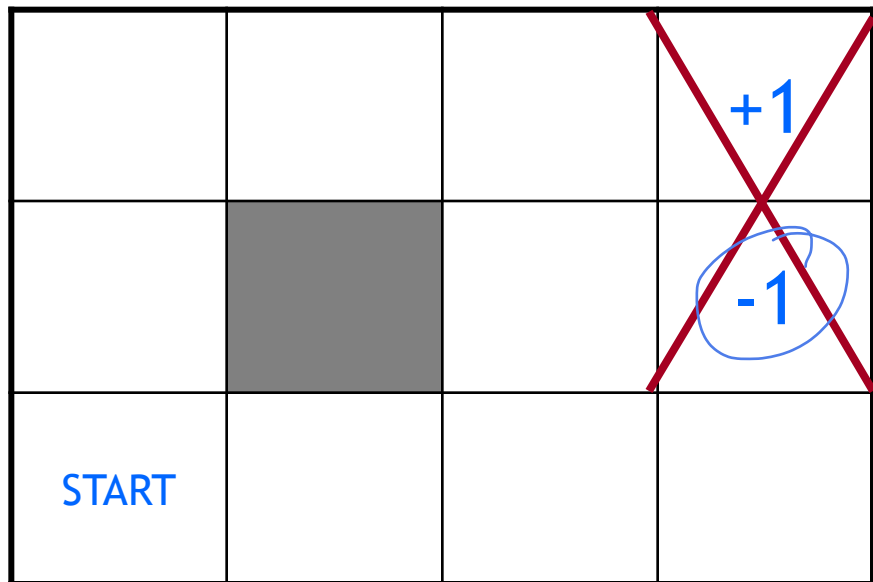
80% move UP

10% move LEFT

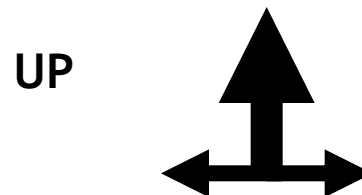
10% move RIGHT

- reward +1 at [4,3], -1 at [4,2]
- reward -0.04 for each step
- what's the strategy to achieve max reward?

# Example: Frozen Lake



actions: UP, DOWN, LEFT, RIGHT



80%

move UP

10%

move LEFT

10%

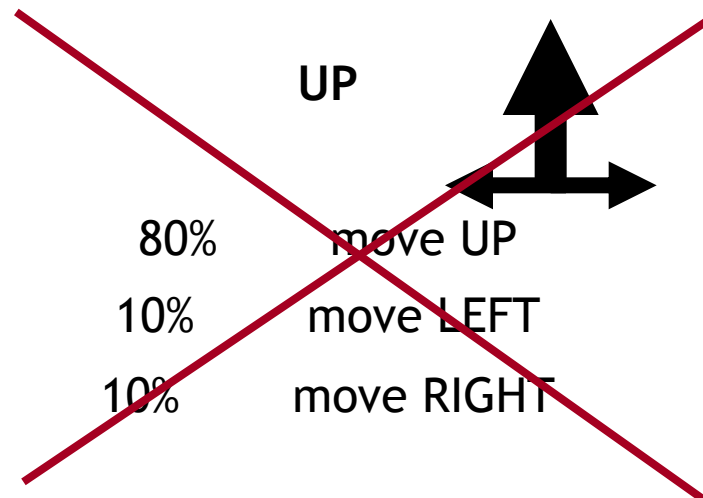
move RIGHT

- ~~reward +1 at [4,3], -1 at [4,2]~~
- ~~reward -0.04 for each step~~
- what's the strategy to achieve max reward?

# Example: Frozen Lake

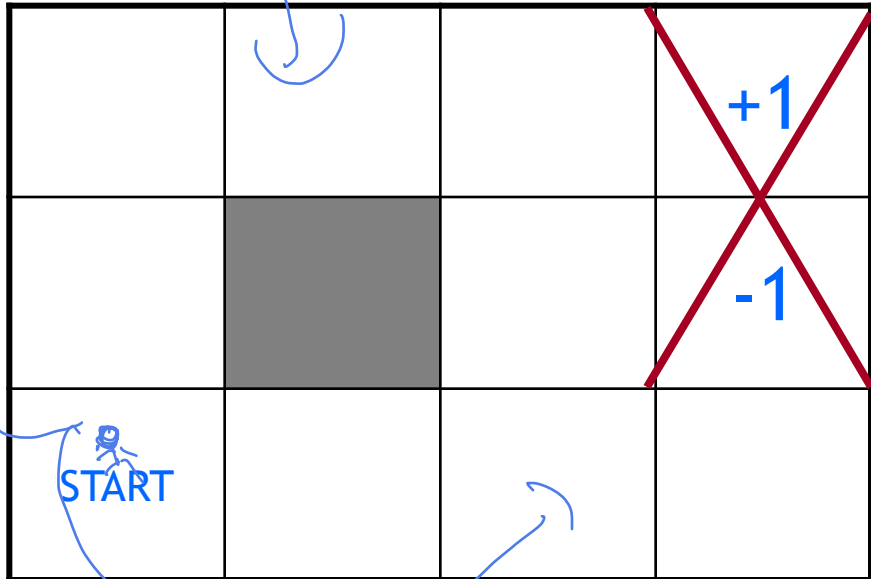
			<del>+1</del>
			<del>-1</del>
START			

actions: UP, DOWN, LEFT, RIGHT

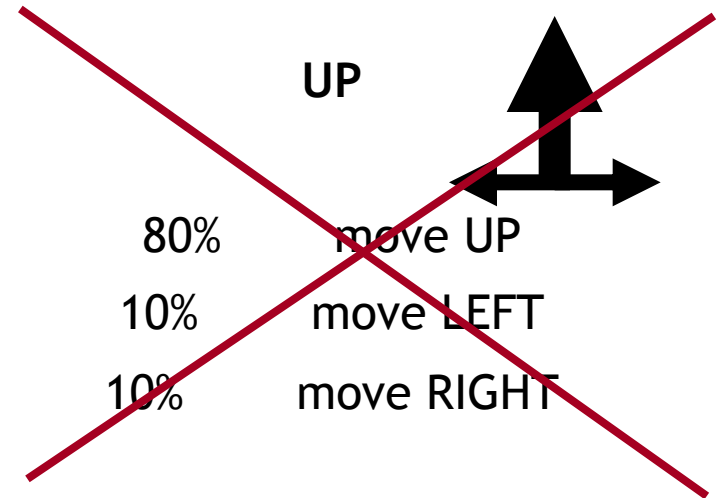


- ~~reward +1 at [4,3], -1 at [4,2]~~
- ~~reward -0.04 for each step~~
- what's the strategy to achieve max reward?

# Example: Frozen Lake



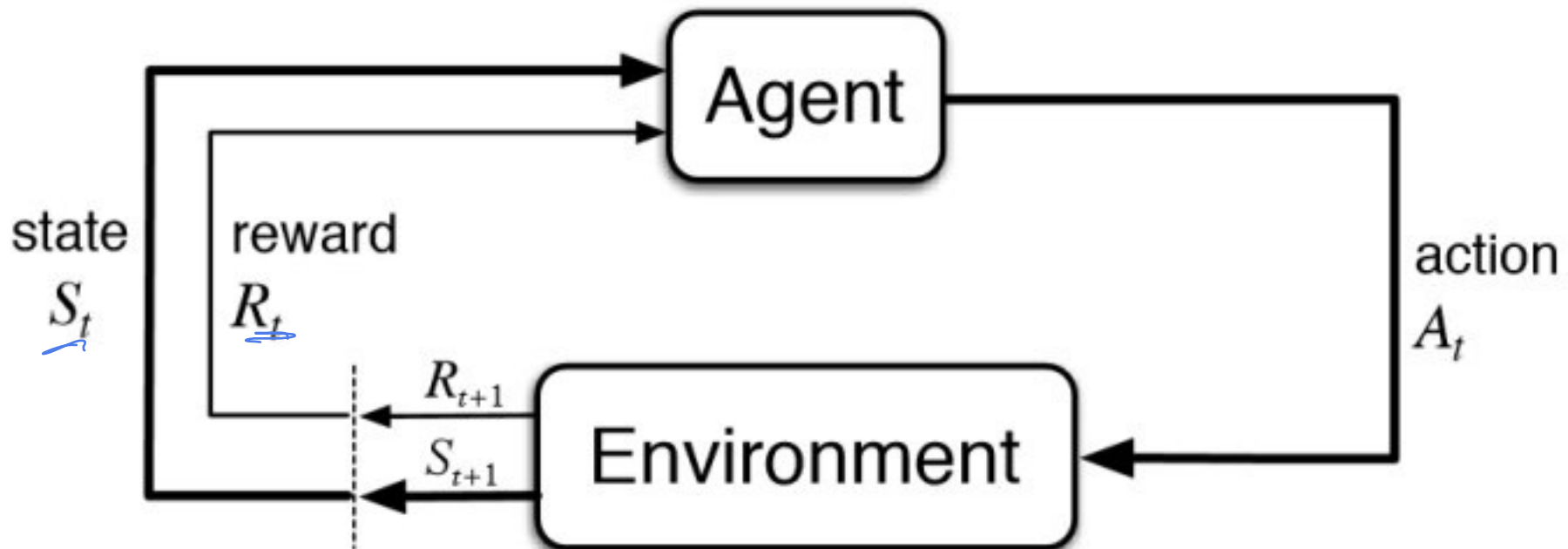
**Action 1, Action 2, Action 3, Action 4**  
actions. ~~UP, DOWN, LEFT, RIGHT~~



- ~~reward +1 at [4,3], -1 at [4,2]~~
- ~~reward -0.04 for each step~~
- what's the strategy to achieve max reward?

# Instead, reinforcement learning agents have “online” access to an environment

- State, Action, Reward
- Unknown reward function, unknown state-transitions.
- Agents can “act” and “experiment”, rather than only doing offline planning.



# Idea 1: Model-based Reinforcement Learning

- Model-based idea
  - Let's approximate the model based on experiences
  - Then solve for the values as if the learned model were correct
- Step 1: Get data by running the agent to explore
  - Many data points of the form:  
 $\{(s_1, a_1, s_2, r_1), \dots, (s_N, a_N, s_{N+1}, r_N)\}$
- Step 2: Estimate the model parameters
  - $\hat{P}(s'|s, a)$  --- again this is a CPT we need to observe the transition many times for each  $s, a$
  - $\hat{r}(s', a, s)$  --- this is an estimate of the empirical rewards.



Then we can plug in these estimates and then use dynamic programming for policy evaluation / improvements.

$$V_{k+1}^{\pi}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \hat{P}(s'|s, a) [\hat{r}(s, a, s') + \gamma V_k^{\pi}(s')]$$

$$\pi' \leftarrow \arg \max_a \sum_{s'} \hat{P}(s'|s, a) [\hat{r}(s, a, s') + \gamma V_k^{\pi}(s')]$$

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} \hat{P}(s'|s, a) [\hat{r}(s, a, s') + \gamma V_k(s')]$$

Then we can plug in these estimates and then use dynamic programming for policy evaluation / improvements.

$$V_{k+1}^{\pi}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \hat{P}(s'|s, a) [\hat{r}(s, a, s') + \gamma V_k^{\pi}(s')]$$

$$\pi' \leftarrow \arg \max_a \sum_{s'} \hat{P}(s'|s, a) [\hat{r}(s, a, s') + \gamma V_k^{\pi}(s')]$$

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} \hat{P}(s'|s, a) [\hat{r}(s, a, s') + \gamma V_k(s')]$$

Then we can plug in these estimates and then use dynamic programming for policy evaluation / improvements.

Policy Evaluation

$$V_{k+1}^\pi(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \hat{P}(s'|s, a) [\hat{r}(s, a, s') + \gamma V_k^\pi(s')]$$

$$\pi' \leftarrow \arg \max_a \sum_{s'} \hat{P}(s'|s, a) [\hat{r}(s, a, s') + \gamma V_k^\pi(s')]$$

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} \hat{P}(s'|s, a) [\hat{r}(s, a, s') + \gamma V_k(s')]$$

\* Note the “hat”. Usually it indicates empirical estimates.

if  $\hat{P} \approx P$  for  $\hat{r} \approx r$   
 $\Rightarrow V_{k+1} \approx V_k$

Then we can plug in these estimates and then use dynamic programming for policy evaluation / improvements.

$$V_{k+1}^{\pi}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \hat{P}(s'|s, a) [\hat{r}(s, a, s') + \gamma V_k^{\pi}(s')]$$

$$\pi' \leftarrow \arg \max_a \sum_{s'} \hat{P}(s'|s, a) [\hat{r}(s, a, s') + \gamma V_k^{\pi}(s')]$$

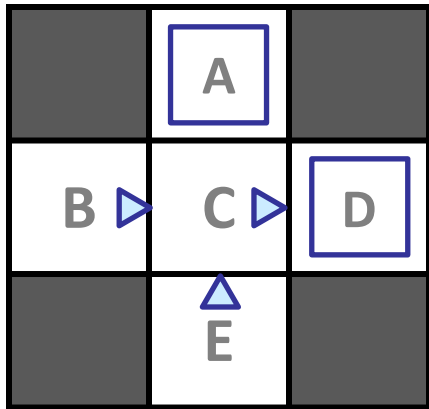
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} \hat{P}(s'|s, a) [\hat{r}(s, a, s') + \gamma V_k(s')]$$

\* Note the “**hat**”. Usually it indicates empirical estimates.

\* These iterations will produce  $\hat{V}^*$  and  $\hat{Q}^*$  functions, and then  $\hat{\pi}^*$

# Example: Model-Based RL (2 min exercise)

Input Policy  $\pi$



Assume:  $\gamma = 1$

Observed Episodes (Training)

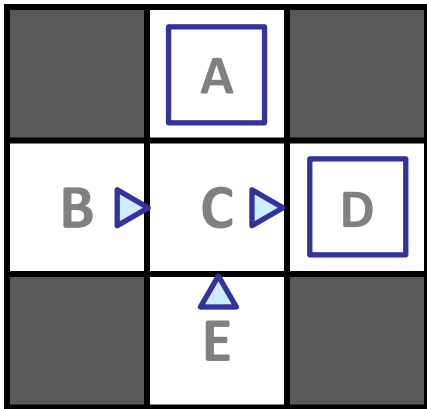
Learned Model

$$\hat{P}(s'|s, a)$$

$$\hat{r}(s, a, s')$$

# Example: Model-Based RL (2 min exercise)

Input Policy  $\pi$



Assume:  $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1  
 C, east, D, -1  
 D, exit, x, +10

Episode 2

B, east, C, -1  
 C, east, D, -1  
 D, exit, x, +10

Episode 3

E, north, C, -1  
 C, east, D, -1  
 D, exit, x, +10

Episode 4

E, north, C, -1  
 C, east, A, -1  
 A, exit, x, -10

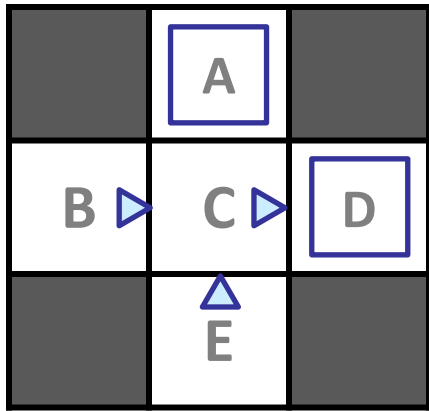
Learned Model

$$\hat{P}(s'|s, a)$$

$$\hat{r}(s, a, s')$$

# Example: Model-Based RL (2 min exercise)

Input Policy  $\pi$



Assume:  $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1

C, east, D, -1

D, exit, x, +10

Episode 2

B, east, C, -1

C, east, D, -1

D, exit, x, +10

Episode 3

E, north, C, -1

C, east, D, -1

D, exit, x, +10

-50

Episode 4

E, north, C, -1

C, east, A, -1

A, exit, x, -10

$T(B, \text{east}, C)$   
 $= P(C | B, \text{east})$   
 Learned Model

$\hat{P}(s' | s, a)$

$T(B, \text{east}, C) = \frac{2}{2} = 1$

$T(C, \text{east}, D) = \frac{2}{4}$

$T(C, \text{east}, A) = \frac{1}{4}$

...

$\hat{r}(s, a, s')$

$R(B, \text{east}, C) = -1$

$R(C, \text{east}, D) = -1$

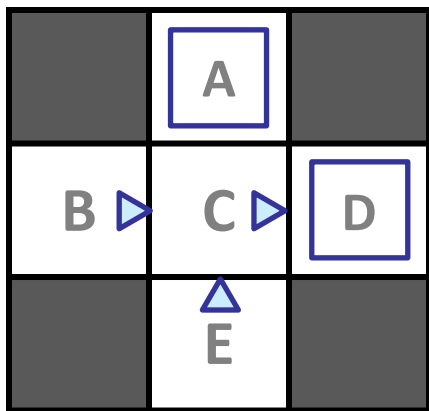
$R(D, \text{exit}, x) = +10$

...

10 transition

# Example: Model-Based RL (2 min exercise)

Input Policy  $\pi$



Assume:  $\gamma = 1$

Observed Episodes (Training)

Learned Model

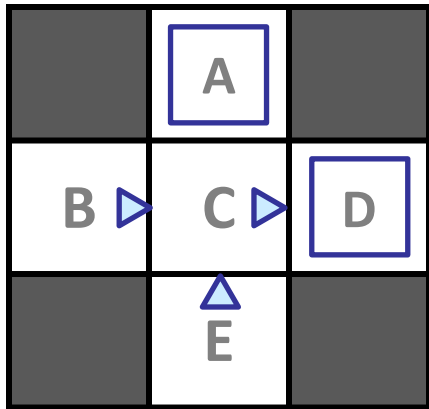
$$\hat{P}(s'|s, a)$$

$$\hat{r}(s, a, s')$$



# Example: Model-Based RL (2 min exercise)

Input Policy  $\pi$



Assume:  $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 2

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 3

E, north, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 4

E, north, C, -1  
C, east, A, -1  
A, exit, x, -10

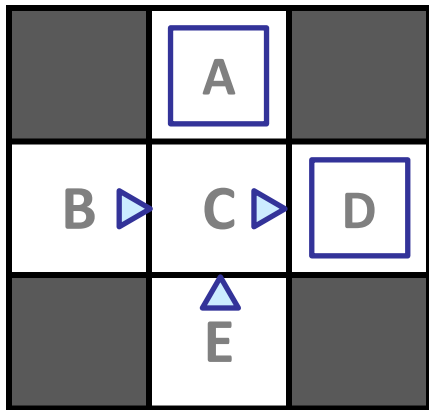
Learned Model

$$\hat{P}(s'|s, a)$$

$$\hat{r}(s, a, s')$$

# Example: Model-Based RL (2 min exercise)

Input Policy  $\pi$



Assume:  $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1  
 C, east, D, -1  
 D, exit, x, +10

Episode 2

B, east, C, -1  
 C, east, D, -1  
 D, exit, x, +10

Episode 3

E, north, C, -1  
 C, east, D, -1  
 D, exit, x, +10

Episode 4

E, north, C, -1  
 C, east, A, -1  
 A, exit, x, -10

Learned Model

$$\hat{P}(s'|s, a)$$

T(B, east, C) = 1.00  
 T(C, east, D) = 0.75  
 T(C, east, A) = 0.25  
 ...

$$\hat{r}(s, a, s')$$

R(B, east, C) = -1  
 R(C, east, D) = -1  
 R(D, exit, x) = +10  
 ...

This is simply the “Exploration-First” strategy! But there are complications.

# This is simply the “Exploration-First” strategy! But there are complications.

- In bandits problems
  - Uniformly sample the actions for  $N$  rounds.
  - Guarantees that each choice is explored  $O(N/k)$  times.

# This is simply the “Exploration-First” strategy! But there are complications.

- In bandits problems
  - Uniformly sample the actions for  $N$  rounds.
  - Guarantees that each choice is explored  $O(N/k)$  times.
- For MDPs
  - Often we need to take a carefully chosen sequence of actions to reach a state
  - The chance of randomly running into a state can be **exponentially small**.

# This is simply the “Exploration-First” strategy! But there are complications.

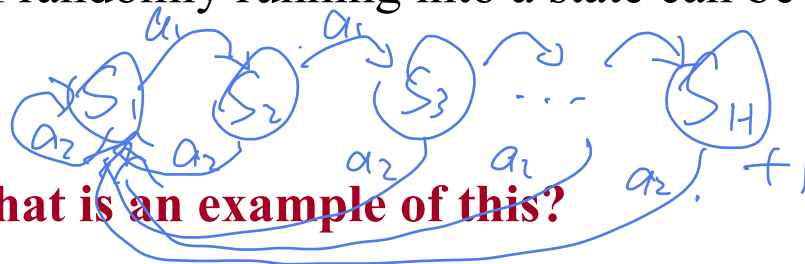
- In bandits problems
  - Uniformly sample the actions for  $N$  rounds.
  - Guarantees that each choice is explored  $O(N/k)$  times.
- For MDPs
  - Often we need to take a carefully chosen sequence of actions to reach a state
  - The chance of randomly running into a state can be **exponentially small**.
  - **Question: What is an example of this?**

# This is simply the “Exploration-First” strategy! But there are complications.

- In bandits problems
  - Uniformly sample the actions for N rounds.
  - Guarantees that each choice is explored  $O(N/k)$  times.
- For MDPs
  - Often we need to take a carefully chosen sequence of actions to reach a state

- The chance of randomly running into a state can be **exponentially small.**

- **Question: What is an example of this?**



Handwritten notes:  $P(S_H)$  and  $\approx O(e^{-H})$ .

\*Need to somehow update the “exploration policy” on the fly!

# More caveats

- 

?



# More caveats

- The fitted model is just an approximation of the environment.

- 

?

## More caveats

- The fitted model is just an approximation of the environment.
- How does the error in the fitted MDP translate into the error in the estimated value functions  $V^*$  and  $Q^*$ ?

- 

?

## More caveats

- $Q^*$  ?
- The fitted model is just an approximation of the environment.
- How does the error in the fitted MDP translate into the error in the estimated value functions  $V^*$  and  $Q^*$ ?
- How does the error in the estimated  $Q^*$  function affect the suboptimality of the policy that maximizes  $Q^*$  ?  
 $Q^*$  ?

# More caveats

- $Q^*$  ?
- The fitted model is just an approximation of the environment.
- How does the error in the fitted MDP translate into the error in the estimated value functions  $V^*$  and  $Q^*$ ?
- How does the error in the estimated  $Q^*$  function affect the suboptimality of the policy that maximizes  $Q^*$  ?

$Q^*$  ?

$$\underline{Q^*(\hat{M})} \approx \underline{Q^*(M)}$$

- Answered by “Simulation Lemma” (Kearns and Singh, 2002)

- Resurgence of research on this more recently: Yin and W. (2020),  
Yin, Bai and W. (2020)

$$\underline{Q} \approx Q^* \text{ then } \underline{Q} \approx Q^*$$

## Even more caveats

- How many free parameters are there to represent an MDP?

## Even more caveats

$$P(s'|s,A)$$

- How many free parameters are there to represent an MDP?
  - Ans:  $O(S^2A)$

## Even more caveats

- How many free parameters are there to represent an MDP?
  - Ans:  $O(S^2A)$
- $S$  is often large
  - 9-puzzle, Tic-Tac-Toe:  $9! = 362,800$ ,  $S^2 = 1.3 \cdot 10^{11}$

# Even more caveats

- How many free parameters are there to represent an MDP?
  - Ans:  $O(S^2A)$
- $S$  is often large
  - 9-puzzle, Tic-Tac-Toe:  $9! = 362,800$ ,  $S^2 = 1.3 \cdot 10^{11}$
  - PACMAN with 20 by 20 grid.  $S = O(2^{400})$ ,  $S^2 = O(2^{800})$

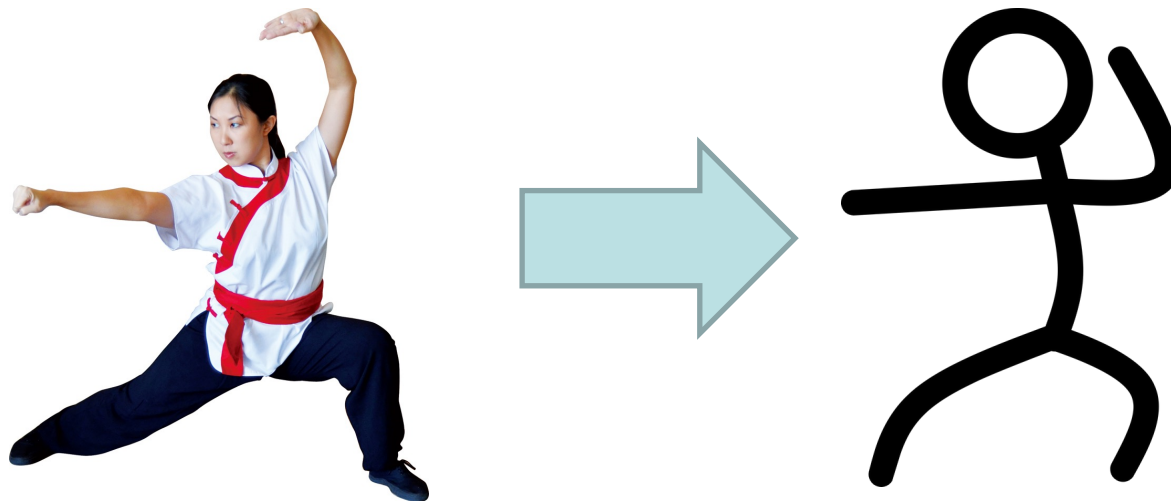


## Even more caveats

- How many free parameters are there to represent an MDP?
  - Ans:  $O(S^2A)$
- $S$  is often large
  - 9-puzzle, Tic-Tac-Toe:  $9! = 362,800$ ,  $S^2 = 1.3 \cdot 10^{11}$
  - PACMAN with 20 by 20 grid.  $S = O(2^{400})$ ,  $S^2 = O(2^{800})$
- In practice, we often have to use an approximate model.

## Even more caveats

- How many free parameters are there to represent an MDP?
  - Ans:  $O(S^2A)$
- $S$  is often large
  - 9-puzzle, Tic-Tac-Toe:  $9! = 362,800$ ,  $S^2 = 1.3 \cdot 10^{11}$
  - PACMAN with 20 by 20 grid.  $S = O(2^{400})$ ,  $S^2 = O(2^{800})$
- In practice, we often have to use an approximate model.



## Idea 2: **Model-free** Reinforcement Learning

- Do we need the model? Can we learn the Q function directly?

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

## Idea 2: Model-free Reinforcement Learning

- Do we need the model? Can we learn the Q function directly?
  - **How many free parameters are there to represent the Q-function?**

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

## Idea 2: Model-free Reinforcement Learning

- Do we need the model? Can we learn the Q function directly?
  - **How many free parameters are there to represent the Q-function?**
  - **Ans:  $SA \ll O(S^2A)$**

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

## Idea 2: Model-free Reinforcement Learning

- Do we need the model? Can we learn the Q function directly?
  - **How many free parameters are there to represent the Q-function?**
  - **Ans: SA  $\ll$  O(S<sup>2</sup>A)**
- Recall: Policy iterations

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

## Idea 2: Model-free Reinforcement Learning

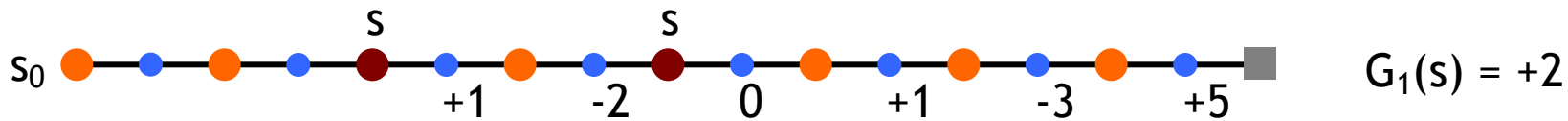
- Do we need the model? Can we learn the Q function directly?
  - **How many free parameters are there to represent the Q-function?**
  - **Ans:  $SA \ll O(S^2A)$**
- Recall: Policy iterations

$$\pi_0 \xrightarrow{\underline{E}} V^{\pi_0} \xrightarrow{\underline{I}} \pi_1 \xrightarrow{\underline{E}} V^{\pi_1} \xrightarrow{\underline{I}} \dots \xrightarrow{\underline{I}} \pi^* \xrightarrow{\underline{E}} V^*$$

- Maybe we can do policy evaluation without estimating the model?

# Monte Carlo Policy Evaluation (Prediction)

- want to estimate  $V^\pi(s)$ 
  - = expected return starting from  $s$  and following  $\pi$
  - estimate as average of observed returns in state  $s$
- We can execute the policy  $\pi$
- first-visit MC
  - average returns following the first visit to state  $s$



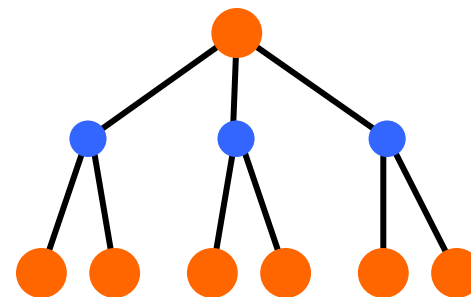




# Monte Carlo Policy Optimization (Control)

- $V^\pi$  not enough for policy improvement
  - need exact model of environment

- estimate  $Q^\pi(s,a)$   
 $\pi'(s) = \arg \max_a Q^\pi(s, a)$



- MC control

$$\pi_0 \xrightarrow{E} Q^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} Q^{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \xrightarrow{E} Q^*$$

- update after each episode

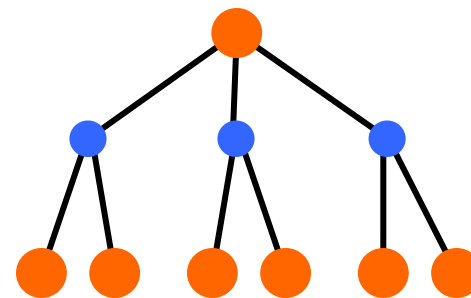
- Two problems

- greedy policy won't explore all actions
- Requires many independent episodes for the estimated value function to be accurate.

# Monte Carlo Policy Optimization (Control)

- $V^\pi$  not enough for policy improvement
  - need exact model of environment

- estimate  $Q^\pi(s,a)$   
 $\pi'(s) = \arg \max_a Q^\pi(s, a)$



- MC control

$$\pi_0 \xrightarrow{E} Q^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} Q^{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \xrightarrow{E} Q^*$$

- update after each episode

- Two problems

- greedy policy won't explore all actions **eps-greedy!**
- Requires many independent episodes for the estimated value function to be accurate.

# Improved Monte-Carlo Q-function estimate using Bellman equations

- Recall:

$$Q^\pi(s, a) = \sum_{s'} P(s'|s, a)[r(s, a, s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s', a')]$$

$$Q^\pi(s, a) = r^\pi(s, a) + \gamma \mathbb{E}_{s' \sim P(s'|s, a)} [V^\pi(s')]$$

# Improved Monte-Carlo Q-function estimate using Bellman equations

- Recall:

$$Q^\pi(s, a) = \sum_{s'} P(s'|s, a)[r(s, a, s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s', a')]$$

$$Q^\pi(s, a) = r^\pi(s, a) + \gamma \mathbb{E}_{s' \sim P(s'|s, a)}[V^\pi(s')]$$

- We can use the empirical (Monte Carlo) estimate.

$$\widehat{Q}^\pi(s, a) = \widehat{r}^\pi(s, a) + \gamma \widehat{\mathbb{E}}_{s' \sim P(s'|s, a)}[\widehat{V}^\pi(s')]$$

# Improved Monte-Carlo Q-function estimate using Bellman equations

- Recall:

$$Q^\pi(s, a) = \sum_{s'} P(s'|s, a)[r(s, a, s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s', a')]$$

$$Q^\pi(s, a) = r^\pi(s, a) + \gamma \mathbb{E}_{s' \sim P(s'|s, a)} [V^\pi(s')]$$

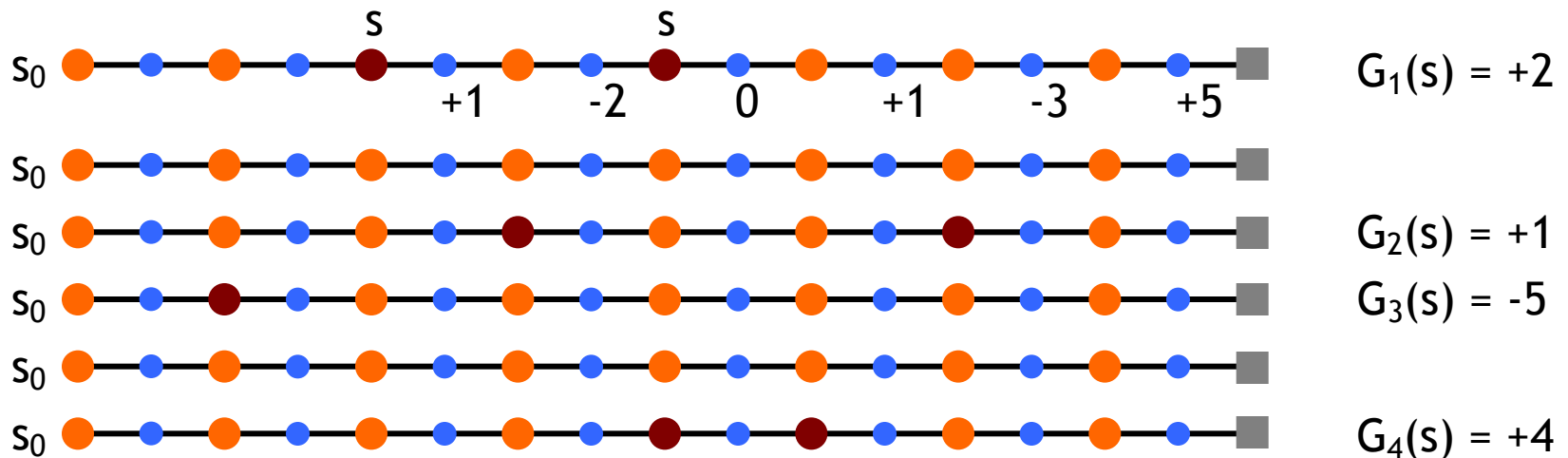
- We can use the empirical (Monte Carlo) estimate.

$$\widehat{Q}^\pi(s, a) = \widehat{r}^\pi(s, a) + \gamma \widehat{\mathbb{E}}_{s' \sim P(s'|s, a)} [\widehat{V}^\pi(s')]$$

\*No need to estimate  $P(s' | s, a)$  or  $r(s, a, s')$  as intermediate steps.

\*Require only  $O(SA)$  space, rather than  $O(S^2A)$

# Online averaging representation of MC



$$V^\pi(s) \approx (2 + 1 - 5 + 4)/4 = 0.5$$

- Alternative, *online averaging* update

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)], \quad \text{where } \alpha = 1/N_{S_t}$$

# DP + MC = Temporal Difference Learning

- Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)],$$



# DP + MC = Temporal Difference Learning

- Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)],$$

Issue:  $G_t$  can only be obtained after the entire episode!

# DP + MC = Temporal Difference Learning

- Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)],$$

Issue:  $G_t$  can only be obtained after the entire episode!

- The idea of TD learning:

$$\mathbb{E}_\pi [G_t] = \mathbb{E}_\pi [R_t | S_t] + \gamma V^\pi(S_{t+1})$$

# DP + MC = Temporal Difference Learning

- Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)],$$

Issue:  $G_t$  can only be obtained after the entire episode!

- The idea of TD learning:

$$\mathbb{E}_\pi [G_t] = \mathbb{E}_\pi [R_t | S_t] + \gamma V^\pi(S_{t+1})$$

We only need one step before we can plug-in and estimate the RHS!

# DP + MC = Temporal Difference Learning

- Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)],$$

Issue:  $G_t$  can only be obtained after the entire episode!

- The idea of TD learning:

$$\mathbb{E}_\pi [G_t] = \mathbb{E}_\pi [R_t | S_t] + \gamma V^\pi(S_{t+1})$$

We only need one step before we can plug-in and estimate the RHS!

- TD-Policy evaluation

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

# DP + MC = Temporal Difference Learning

- Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)],$$

Issue:  $G_t$  can only be obtained after the entire episode!

- The idea of TD learning:

$$\mathbb{E}_\pi [G_t] = \mathbb{E}_\pi [R_t | S_t] + \gamma V^\pi(S_{t+1})$$

We only need one step before we can plug-in and estimate the RHS!

- TD-Policy evaluation

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

**Bootstrapping!**

# Bootstrap's origin

- “The Surprising Adventures of Baron Münchhausen”
  - Rudolf Erich Raspe, 1785



**PULL  
YOURSELF  
UP BY  
THE  
BOOT  
STRAPS!!!**



- In statistics: Brad Efron's resampling methods
- In computing: Booting...
- In RL: It simply means TD learning

# TD policy optimization (TD-control )

- SARSA (On-Policy TD-control)

- Update the Q function by bootstrapping Bellman Equation

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

- Choose the next A' using Q, e.g., eps-greedy.

- Q-Learning (Off-policy TD-control)

- Update the Q function by bootstrapping Bellman Optimality Eq.

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

- Choose the next A' using Q, e.g., eps-greedy, or any other policy.

## Remarks:

- These are **proven to converge** asymptotically.
- Much more data-efficient in practice, than MC.
- Regret analysis is still active area of research.

# Advantage of TD over Monte Carlo

- Given a trajectory, a roll-out, of  $T$  steps.
  - MC updates the  $Q$  function only once
  - TD updates the  $Q$  function (and the policy)  $T$  times!



# Advantage of TD over Monte Carlo

- Given a trajectory, a roll-out, of  $T$  steps.
  - MC updates the  $Q$  function only once
  - TD updates the  $Q$  function (and the policy)  $T$  times!

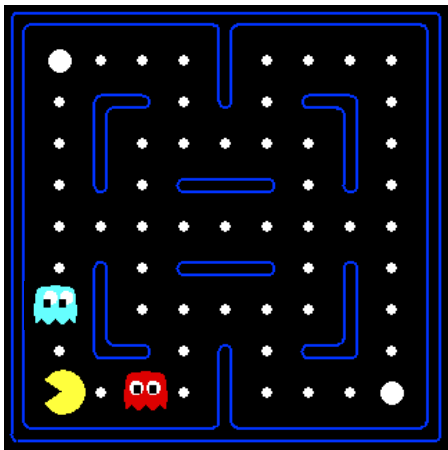
**Remark:** This is the same kind of improvement from Gradient Descent to Stochastic Gradient Descent (SGD).

# The problem of large state-space is still there

- We need to represent and learn SA parameters in Q-learning and SARSA.
- S is often large
  - 9-puzzle, Tic-Tac-Toe:  $9! = 362,800$ ,  $S^2 = 1.3 \cdot 10^{11}$
  - PACMAN with 20 by 20 grid.  $S = O(2^{400})$ ,  $S^2 = O(2^{800})$
- $O(S)$  is not acceptable in some cases.
- Need to think of ways to “generalize”/share information across states.

# Example: Pacman

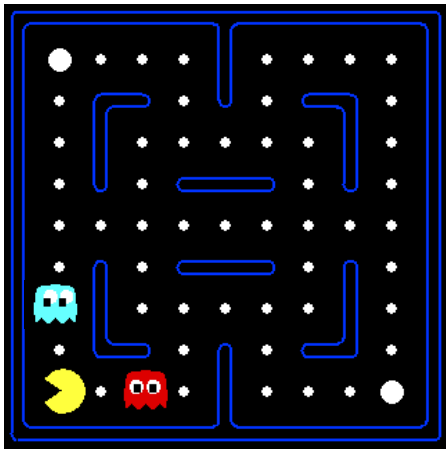
Let's say we discover  
through experience  
that this state is bad:



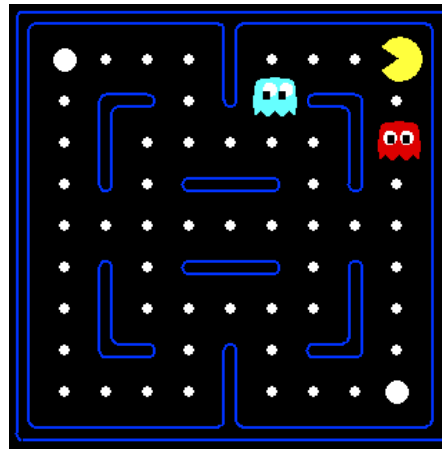
(From Dan Klein and Pieter Abbeel)

# Example: Pacman

Let's say we discover through experience that this state is bad:



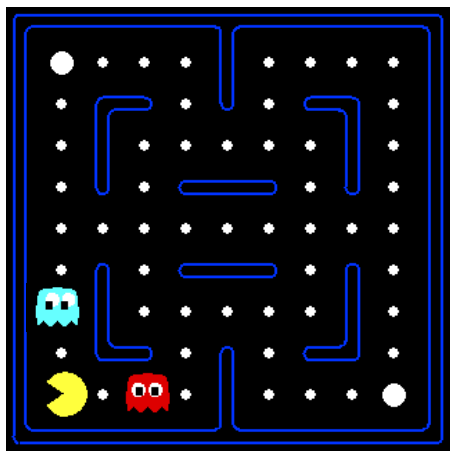
In naïve q-learning, we know nothing about this state:



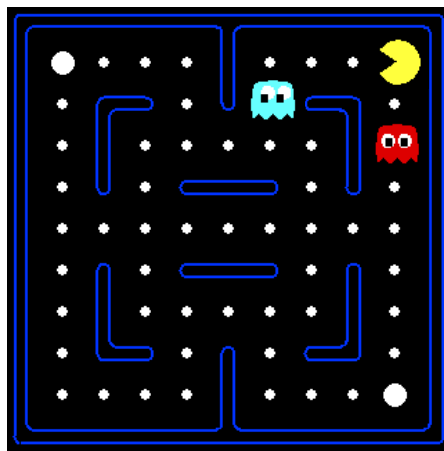
(From Dan Klein and Pieter Abbeel)

# Example: Pacman

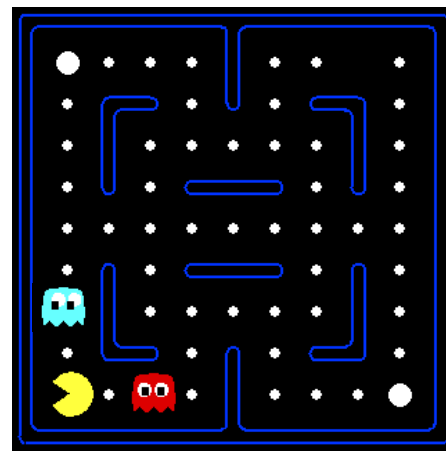
Let's say we discover through experience that this state is bad:



In naïve q-learning, we know nothing about this state:



Or even this one!



(From Dan Klein and Pieter Abbeel)

# Video of Demo Q-Learning Pacman – Tiny – Watch All



# Video of Demo Q-Learning Pacman – Tiny – Silent Train



# Video of Demo Q-Learning Pacman – Tricky – Watch All

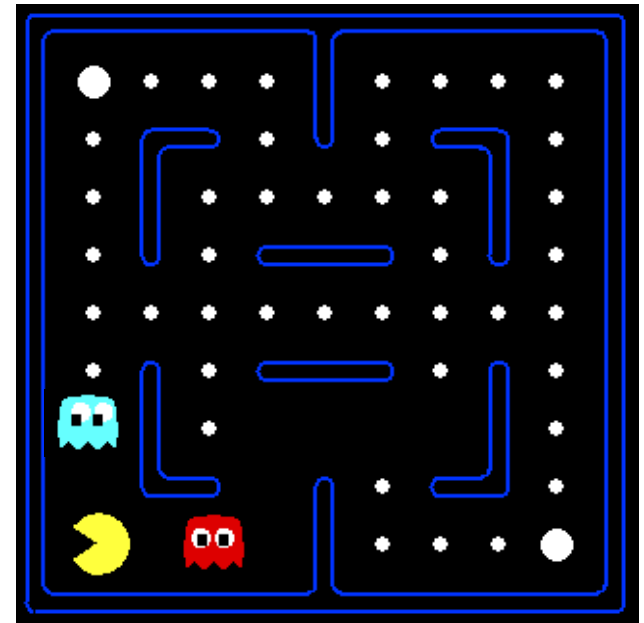




# Why not use an evaluation function?

## A Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - $1 / (\text{dist to dot})^2$
    - Is Pacman in a tunnel? (0/1)
    - ..... etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



# Linear Value Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:
  - $V_{\mathbf{w}}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$
  - $Q_{\mathbf{w}}(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \dots + w_n f_n(s,a)$
- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but actually be very different in value!

# Updating a linear value function

# Updating a linear value function

- Original Q learning rule tries to reduce prediction error at  $s$ ,  $a$ :

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) ]$$

# Updating a linear value function

- Original Q learning rule tries to reduce prediction error at s, a:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) ]$$

- Instead, we update the weights to try to reduce the error at s, a:

# Updating a linear value function

- Original Q learning rule tries to reduce prediction error at  $s, a$ :

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

- Instead, we update the weights to try to reduce the error at  $s, a$ :

$$\begin{aligned} w_i &\leftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \leftarrow \frac{\partial Q_w(s,a)}{\partial w_i} \\ &= w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] f_i(s,a) \end{aligned}$$

# Updating a linear value function

- Original Q learning rule tries to reduce prediction error at  $s, a$ :

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) ]$$

- Instead, we update the weights to try to reduce the error at  $s, a$ :

$$\begin{aligned} w_i &\leftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) ] \partial Q_w(s,a) / \partial w_i \\ &= w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) ] f_i(s,a) \end{aligned}$$

- Qualitative justification:

- Pleasant surprise: increase weights on positive features, decrease on negative ones
- Unpleasant surprise: decrease weights on positive features, increase on negative ones

# Q-Learning with function approximation

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$



# Q-Learning with function approximation

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Q-learning with linear Q-functions:

# Q-Learning with function approximation

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Q-learning with linear Q-functions:

$$\text{transition} = (s, a, r, s')$$

# Q-Learning with function approximation

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Q-learning with linear Q-functions:

transition =  $(s, a, r, s')$

difference =  $\left[ r + \gamma \max_{a'} Q(s', a') \right] - Q(s, a)$

# Q-Learning with function approximation

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Q-learning with linear Q-functions:

$$\text{transition} = (s, a, r, s')$$

$$\text{difference} = \left[ r + \gamma \max_{a'} Q(s', a') \right] - Q(s, a)$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha [\text{difference}] \quad \text{Exact Q's}$$

# Q-Learning with function approximation

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Q-learning with linear Q-functions:

$$\text{transition} = (s, a, r, s')$$

$$\text{difference} = \left[ r + \gamma \max_{a'} Q(s', a') \right] - Q(s, a)$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha [\text{difference}] \quad \text{Exact Q's}$$

$$w_i \leftarrow w_i + \alpha [\text{difference}] f_i(s, a) \quad \text{Approximate Q's}$$

# Q-Learning with function approximation

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Q-learning with linear Q-functions:

transition =  $(s, a, r, s')$

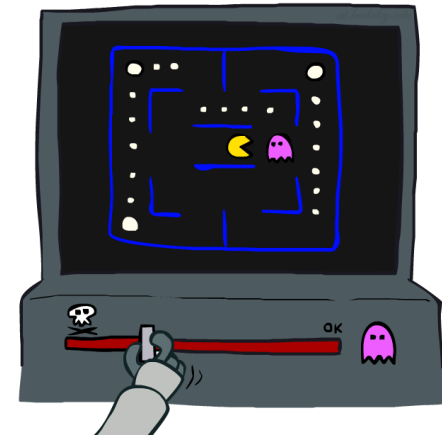
difference =  $\left[ r + \gamma \max_{a'} Q(s', a') \right] - Q(s, a)$

$Q(s, a) \leftarrow Q(s, a) + \alpha [\text{difference}]$       Exact Q's

$w_i \leftarrow w_i + \alpha [\text{difference}] f_i(s, a)$       Approximate Q's

- Intuitive interpretation:

- Adjust weights of active features
- E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features



# Q-Learning with function approximation

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Q-learning with linear Q-functions:

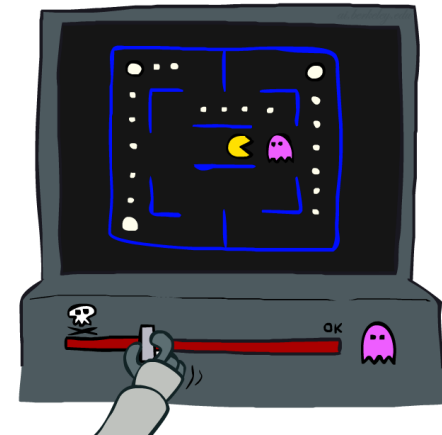
$$\text{transition} = (s, a, r, s')$$

$$\text{difference} = \left[ r + \gamma \max_{a'} Q(s', a') \right] - Q(s, a)$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha [\text{difference}] \quad \text{Exact Q's}$$

$$w_i \leftarrow w_i + \alpha [\text{difference}] f_i(s, a) \quad \text{Approximate Q's}$$

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features
- Formal justification: online least squares (Read the textbook!)



# PACMAN Q-Learning (Linear function approx.)





# So far, in RL algorithms

- Model-based approaches
  - Estimate the MDP parameters.
  - Then use policy-iterations, value iterations.
- Monte Carlo methods:
  - estimating the rewards by empirical averages
- Temporal Difference methods:
  - Combine Monte Carlo methods with Dynamic Programming
- Linear function approximation in Q-learning
  - Similar to SGD
  - Learning heuristic function

# So far, in RL algorithms

- Model-based approaches
  - Estimate the MDP parameters.
  - Then use policy-iterations, value iterations.
- Monte Carlo methods:
  - estimating the rewards by empirical averages
- Temporal Difference methods:
  - Combine Monte Carlo methods with Dynamic Programming
- Linear function approximation in Q-learning
  - Similar to SGD
  - Learning heuristic function

\*Question: What is the policy class  $\Pi$  of interest in these methods?

# Next lecture

- Wrap up RL lectures
  - Policy gradients methods
- Start logic agents / knowledge representation